Subject: Re: How to speed up KRIG2D by 30x
Posted by chris_torrence@NOSPAM on Thu, 10 Oct 2013 17:48:56 GMT
View Forum Message <> Reply to Message

Okay, here is the new code, minus the comments. I added 3 new keywords: DOUBLE, XOUT, YOUT. DOUBLE forces double precision. XOUT, YOUT are output keywords that contain the final X/Y locations. I also vectorized the final loop, for even more speed.

I also changed the C code of GRIDDATA to move the LUSOL out of the loop.

Now, with Mike's changes and the vectorization, I'm getting the following results using Mike's test code at the top with 500 input points and 400 output points:

Old KRIG2D: 30 seconds
New KRIG2D: 0.51 seconds
New GRIDDATA: 1.3 seconds

It's funny that the KRIG2D is faster than GRIDDATA, but that's because GRIDDATA has a lot of other machinery to handle sectors & search ellipses. In fact, if you *do* use sectors or search_ellipse, then GRIDDATA has to revert to the old slow algorithm with the ludc in the inner loop. Oh well.

Anyway, let me know how this code looks. If all goes well, this will make it into IDL 8.3, due out in a month or so.

Mike, thanks again for the patch to the code.

Cheers,
Chris
ExelisVIS

```
; Copyright (c) 1993-2013, Exelis Visual Information Solutions, Inc. All
;       rights reserved. Unauthorized reproduction is prohibited.
;Return Exponential Covariance Fcn
; C(d) = C1 exp(-3 d/A)   if d != 0
;      = C1 + C0  if d == 0
;
FUNCTION Krig_expon, d, t
  COMPILE_OPT idl2, hidden

  r = t[2] * exp((-3./t[0]) * d)
  r[WHERE(d eq 0, /NULL)] = t[1] + t[2]
  return, r
end

;Return Spherical Covariance Fcn
; C(d) = C1 [ 1 - 1.5(d/A) + 0.5(d/A)^3] if d < A
;      = C1 + C0  if d == 0
```

```
;      = 0        if d > A
;
FUNCTION Krig_sphere, d, t
  COMPILE_OPT idl2, hidden

  ; The clipping to < 1 will handle the case d > A, so that v = 0.
  r = d/t[0] < 1
  v = t[2]*(1 - r*(1.5 - 0.5*r*r))
  v[WHERE(r eq 0, /NULL)] = t[1] + t[2]
  return, v
end


;   CT, VIS, Oct 2013: Per Mike's suggestion, speed up the algorithm by
;       moving LUSOL out of the loops, and vectorizing the inner loop.
;       Add DOUBLE, XOUT, YOUT keywords.
;       Fix the spherical covariance computation for d > A.
;
;-
FUNCTION krig2d, z, x, y, $
  DOUBLE=doubleIn, $
  REGULAR = regular, XGRID=xgrid, $
  XVALUES = xvalues, YGRID = ygrid, YVALUES = yvalues, $
  GS = gs, BOUNDS = bounds, NX = nx0, NY = ny0, $
  EXPONENTIAL = exponential, SPHERICAL = spherical, $
  XOUT=xout, YOUT=yout

  compile_opt idl2, hidden
  on_error, 2

  s = size(z)   ;Assume 2D
  nx = s[1]
  ny = s[2]

  reg = keyword_set(regular) or (n_params() eq 1)
  dbl = KEYWORD_SET(doubleIn)

  if n_elements(xgrid) eq 2 then begin
    x = (dbl ? dindgen(nx) : findgen(nx)) * xgrid[1] + xgrid[0]
    reg = 1
  endif else if n_elements(xvalues) gt 0 then begin
    if n_elements(xvalues) ne nx then $
      message,'Xvalues must have '+string(nx)+' elements.'
    x = xvalues
    reg = 1
  endif

  if n_elements(ygrid) eq 2 then begin
```

```
  y = (dbl ? dindgen(ny) : findgen(ny)) * ygrid[1] + ygrid[0]
  reg = 1
endif else if n_elements(yvalues) gt 0 then begin
  if n_elements(yvalues) ne ny then $
    message,'Yvalues must have '+string(ny)+' elements.'
  y = yvalues
  reg = 1
endif

if reg then begin
  if s[0] ne 2 then message,'Z array must be 2D for regular grids'
  if n_elements(x) ne nx then x = findgen(nx)
  if n_elements(y) ne ny then y = findgen(ny)
  x = x # replicate(dbl ? 1d : 1., ny)  ;Expand to full arrays.
  y = replicate(dbl ? 1d : 1.,nx) # y
endif

n = n_elements(x)
if n ne n_elements(y) or n ne n_elements(z) then $
  message,'x, y, and z must have same number of elements.'

if keyword_set(exponential) then begin       ;Get model params
  t = exponential
  fname = 'KRIG_EXPON'
endif else if keyword_set(spherical) then begin
  t = spherical
  fname = 'KRIG_SPHERE'
endif else MESSAGE,'Either EXPONENTIAL or SPHERICAL model must be selected.'

if n_elements(t) eq 2 then begin    ;Default value for variance?
  mz = total(z) / n   ;Mean of z
  var = total((z - mz)^2, DOUBLE=dbl)/n ;Variance of Z
  t = [t, var-t[1]] ;Default value for C1
endif

m = n + 1     ;# of eqns to solve
a = dbl ? dblarr(m, m) : fltarr(m, m)

; Construct the symmetric distance matrix.
for i=0, n-2 do begin
  j = [i:n-1]  ; do the columns all at once
  d = (x[i]-x[j])^2 + (y[i]-y[j])^2  ;Distance squared
  ;symmetric
  a[i,j] = d
  a[j,i] = d
endfor

a = call_function(fname, sqrt(a), t)        ;Get coefficient matrix
```

```
  a[n,*] = 1.0          ;Fill edges
  a[*,n] = 1.0
  a[n,n] = 0.0

  LUDC, a, indx, DOUBLE=dbl          ;Solution using LU decomposition

  if n_elements(nx0) le 0 then nx0 = 26 ;Defaults for nx and ny
  if n_elements(ny0) le 0 then ny0 = 26

  xmin = min(x, max = xmax)   ;Make the grid...
  ymin = min(y, max = ymax)

  if n_elements(bounds) lt 4 then bounds = [xmin, ymin, xmax, ymax]

  if n_elements(gs) lt 2 then $   ;Compute grid spacing from bounds
    gs = [(bounds[2]-bounds[0])/(nx0-1.), (bounds[3]-bounds[1])/(ny0-1.)]

  ; Subtract off a fudge factor to lessen roundoff errors.
  nx = ceil((bounds[2]-bounds[0])/gs[0] - 1e-5)+1 ;# of elements
  ny = ceil((bounds[3]-bounds[1])/gs[1] - 1e-5)+1

  xout = bounds[0] + gs[0]*(dbl ? DINDGEN(nx) : FINDGEN(nx))
  yout = bounds[1] + gs[1]*(dbl ? DINDGEN(ny) : FINDGEN(ny))

  ; One extra for Lagrange constraint
  d = dbl ? DBLARR(m) : FLTARR(m)
  result = dbl ? DBLARR(nx,ny,/nozero) : FLTARR(nx,ny,/nozero)

  az = LUSOL(a, indx, [REFORM(z,N_ELEMENTS(z)),0.0], DOUBLE=dbl)
  az = REBIN(TRANSPOSE(az), nx, n+1)
  xx = (SIZE(x, /N_DIM) eq 2) ? x : REBIN(TRANSPOSE(x), nx, n)
  dxsquare = (xx - REBIN(xout, nx, n))^2
  yy = (SIZE(y, /N_DIM) eq 2) ? y : REBIN(TRANSPOSE(y), nx, n)

  ; Do each row separately
  for j=0,ny-1 do begin
    y0 = yout[j]
    ; Do all of the columns in parallel
    d = sqrt(dxsquare + (yy - y0)^2)
    d = CALL_FUNCTION(fname, d, t)
    ; Be sure to add the last row of AZ, which is the Lagrange constraint
    result[*,j] = TOTAL(d*az, 2) + az[*,n]
  endfor

  return, result
end
```